*DOE21*_Python
u03-python-project

Syfte

Syftet med denna uppgift är dels att bekanta dig med teorin i kursen, samt dels att visa dina praktiska färdigheter i Python. Uppgiften har två delar:

Redogörelse (Teoretisk del) - Här skriver du en redogörelse som tar upp det grundläggande kring kunskapsmålen i kursen.

1. om Pythons, syntax, funktioner, variabler och datatyper
2. om användningstillämpningar av Python som skriptspråk i t.ex. systemautomation, skalskript, inbyggda system eller webserverskript

Pythonapplikation (Praktisk del) - I denna del kodar du ihop en gästboksapplikation i Python. Uppgiften behöver använda sig av minst en Class och minst en funktion.

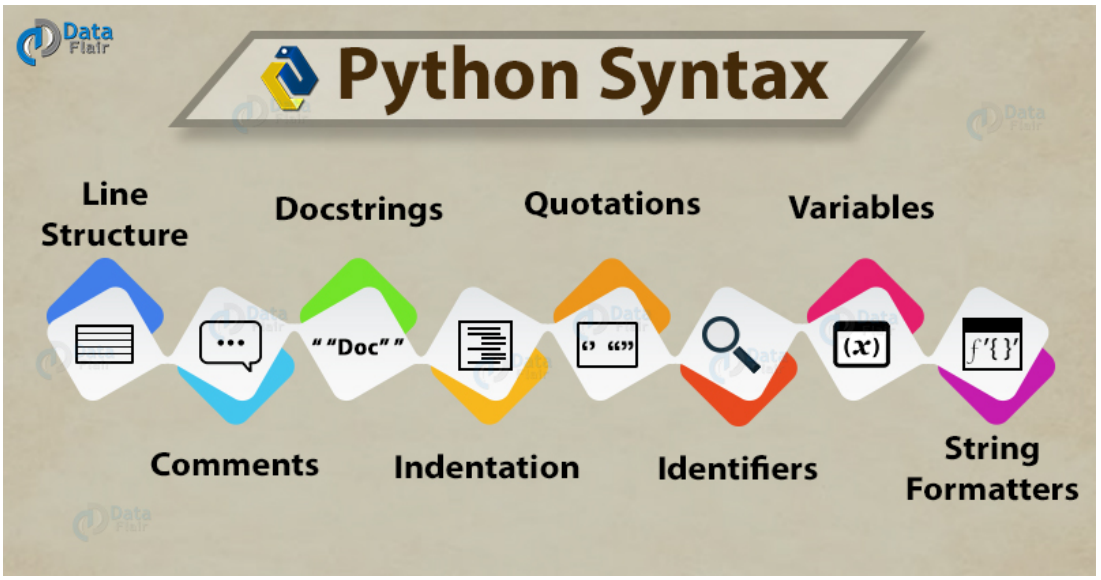
Detta dokument behandlar den teoretiska delen-**Redogörelse.**

Innehållsförteckning:

Python

1. Syntax

Syntax är en uppsättning regler som förklarar Pythons struktur. Reglerna definierar dels hur man skriver ett program (användning av symboler, skiljetecken, ord och hur du ska använda dem) på just det språket samt dels hur den som tolkar reglerna ska förstå koden. Nedan bild visar de vanligaste syntaxer i Python. Några av dessa är kortfattat beskrivna nedan.



<u>Identifierare:</u>
Används för att identifiera en funktion, modul, klass, variabel eller något annat objekt.
Det finns vissa regler man måste följa för att skapa identifierare till exempel att::

- Identifieraren kan börja med en bokstav: AZ eller az.
- Det kan börja med understreck.
- Det kan inte börja med ett nummer eller något annat specialtecken

Python har också regler som hjälper till att beskriva variabeln till exempel att:

- När du namnger en klassvariabel börjar variabeln med en stor bokstav följt av alla små bokstäver.

Linjestruktur:

Python använder sig av logiska linjer där varje rad betyder ett nytt påstående. Tomma rader ignoreras av Python. I nedan två exempel kan ses exempel på hur python läser påståenden

```
# This is an example of how to do it!

print("Dont you just LOVE diet coke!!")
```

output

```
Dont you just LOVE diet coke!!
```

Ovan visar ett exempel på hur python behöver att kod skrivs för att det ska kunna läsas.

Nedan visar att python inte kan läsa när ett påstående byter rad. För att skriva på flera rader krävs ytterligare åtgärder.

```
# This is an example of how NOT to do!

print("Dont you just LOVE
diet coke!!")
```

output

```
SyntaxError: EOL while scanning string literal
```

Indrag:

Andra Programmeringsspråk (C, Java och C++) använder sig av parenteser för att visa kodblock. Det gör du inte i Python utan där anger detta med indrag. Efter varje villkorssats finns ett kolon och nästa rad har en indragning. Grundregeln är att det måste finnas minst ett mellansteg vid indrag, alla påståenden inom kod-blocket måste dras in lika mycket.

Se nedan exempel:

```
num=int(input("Vänligen skriv in ett nummer: \n"))
sum=0
for i in range(0,num):
    sum=sum+i
print("Summan är: ",sum)
```

output

```
Vänligen skriv in ett nummer:
7
Summan är:  21
```

Källor:
https://data-flair.training/blogs/python-syntax-semantics/
https://pythongeeks.org/python-syntax/
https://www.tutorialspoint.com/python/python_basic_syntax.htm

2. Funktioner

En funktion tillämpas när du vill göra något upprepade gånger genom att du anropa den funktionen när du behöver den.

Det finns tre typer av funktioner i Python:

- Inbyggda funktioner, tex. **help()** för att be om hjälp, **min()**att få minimivärdet eller **print()** som skriver ut ett objekt till terminalen,
- Användardefinierade funktioner (UDF), som är funktioner som användare skapar.
- Anonyma funktioner, som även kallas lambdafunktioner. De deklareras inte med standardsökordet def.

En lista över inbyggda funktioner i Python kan ses i länken:
https://docs.python.org/3/library/functions.html

Python Funktionssyntax

- Du måste använda nyckelordet **def,** ge din funktion ett namn, följt av ett par parenteser och sedan avsluta raden med ett kolon (:).
- Om din funktion tar argument, nämns namnen på argumenten (parametrarna) inom den inledande och avslutande parentesen.
- Efter ovan börjar du nästa rad med ett indraget block. Detta är huvuddelen av funktionen som beskriver vad din funktion gör.
- **return** returnerar resultatet av operationen på argumenten.

Observera att varken return eller argument är nödvändiga!

Nedan är ett exempel på hur en funktion kan skrivas utan argument(parametrar) angivet eller användande av ‘return’.

```
def my_func():
  print("Hello! Hope you're doing well in these Covid times!")
my_func()
```

output

```
Hello! Hope you're doing well in these Covid times!
```

Nedan är ett exempel på hur en funktion kan skrivas med argument och ‘return’.

```
# Define a function `sum()`
def sum(a,b):
    return a + b
ab=sum(29, 53)
print(f"The sum is {ab}.")
```

output

```
The sum is 82.
```

Källor:
https://www.freecodecamp.org/news/functions-in-python-a-beginners-guide/
https://www.datacamp.com/community/tutorials/functions-python-tutorial
https://docs.python.org/3/library/functions.html

3. Variabler

En variabel kan beskrivas som ett namn kopplat till ett visst objekt. När du skapar en variabel reserverar du utrymme i minnet. Genom att tilldela olika datatyper till variabler kan du lagra heltal, decimaler eller tecken..För att skapa en variabel tilldelar du den ett värde och börjar sedan använda den. Tilldelningen görs med ett enda likhetstecken (=). Operanden till vänster om operatorn = är namnet på variabeln och operanden till höger om operatorn = är värdet som lagras i variabeln.

Exempel nedan:

```
x = 5
y = "Sara"
print(x)
print(y)
```

output

```
5
Sara
```

Källor:
https://realpython.com/python-variables/
https://www.tutorialspoint.com/python/python_variable_types.htm
https://python-programs.com/python-variables/

4. Datatyper

Olika data innehåller olika slags information. En datatyp är den kategorin som variabeln innehåller. I Python behöver man inte själv ange vilken datatyp en variabel är, detta görs automatiskt.

Det finns olika typer av datatyper.

- **Int:** integer är ett heltal, exempelvis 99, 2, 65, 4.
- **Float:** Ett numeriskt värde med en eller flera decimaler., exempelvis 16.7, 27.5 eller 2.9.
- **String:** En textsträng, exempelvis “Hej Sara”, eller ‘Vad bor du?’.
- **Boolean:** Booleanskt värde. Är antingen True (sant) eller False (falskt). True och False kan även representeras med 1 respektive 0.

Nedan är exempel på hur en integer, float och string kan användas.

```
year = 2022                # An integer assignment
miles_run   = 1000.0       # A floating point
name_of_dog    = "Molly"   # A string

print (year, "is an integer!")
print (miles_run, "is a flout!")
print (name_of_dog, "is a string!")
```

output

```
2022 is an integer!
1000.0 is a flout!
Molly is a string!
```

Källor:
https://www.programmerapython.se/datatyper-och-variabler-i-python/
https://science.nu/courses/data-science-machine-learning-statistik-python/lektion/variabler-datatyper-programmera-python/

Python som skriptspråk i:

5. Systemautomation

Så fort uppgifter är repetitiva är automatisering att föredra.
Ett exempel på när systemautomation används är inom styrning av robotar, “Robotic Process Automation (RPA). Fördelar med Python kontra många andra tillgängliga RPA plattformar är att Python bla. ger stabilitet samt tillåter körning av multipla processer på samma maskin. Det är vanligt att RPA använder sig av Artificiell intelligens (AI) och maskininlärning (ML). Även här är det en fördel för användning av Python då Python ofta är det valda programmeringsspråket i dessa program.

Ett annat användningsområde är att man automatiserar rutinuppgifter som hälsokontroller och säkerhetskopiering av filer med skalskript. När dessa uppgifter blir mer komplexa kan skalskript bli svårare att underhålla. Under 2. Skalskript finns exempel hur man använder python vid skalskript.

Källor:
https://stackabuse.com/executing-shell-commands-with-python/

6. Skalskript

Python kan användas istället för skalskript för automatisering. Python har olika metoder för att köra skalkommandon som ger samma funktionalitet som skalskript. Att köra skalkommandon i Python öppnar möjligheten att automatisera datoruppgifter på ett strukturerat och skalbart sätt.

Metoder som används är tex. os.system, os.propen och subprocess-modellen. **os** står för **operative system**. Den sistnämnda anses vara den rekommenderade modulen pga dess flexibilitet vilket ger användaren tillgång till tex. standardutdata samt standardfel(errors). Den stora skillnaden mellan os.system och os.propen är att os.system INTE kan spara output i en variabel.

Nedan ses några exempel på användning av metoderna.

os.system:

```
import os

os.system("echo This is Sara calling!")

# Use of standard function 'date'
os.system('date')
.....c/Users/aras/u03_python_project/python_scripting$ python3 echo_aras.py
```

output

```
This is Sara calling!
Fri Jan 14 10:51:23 CET 2022
```

Körs ovan i stället i python-terminalen blir output annorlunda. Notera att det nu även skrivs ut en **'0'**. '0'an betyder att körningen gick igenom utan problem(fel/error). Skulle det stå ett annat nummer är det således ett fel/error. Anledningen till detta är att os.system returnerar en "exit" kod som inte är synlig i standardutskriften i konsolen. Det är den när man kör i python-terminalen.

```
>>> import os
>>> os.system('date')
Fri Jan 14 10:52:45 CET 2022
0
>>>
```

För att bekräfta detta lägger jag in ett stavfel i ovan kod. Outputen blir då enligt nedan:

```
>>> import os
>>> os.system('datte')
sh: 1: datte: not found
32512
>>>
```

Notera exit-koden på **32512**.

os.popen:
os.popen returnerar ett filobjekt och för att läsa värdet använder man sig av en "read method".

Se nedan ett enkelt exempel på användning av os.popen.

```
>>> import os
>>> output=os.popen('date')
>>> output.read()
'Fri Jan 14 13:42:00 CET 2022\n'
```

os.popen kan bara läsa in från ett objekt en gång vilket betyder att om vi kör output.read() en gång till i pythonterminalen så får vi nedan resultat.

```
>>> output.read()
''
```

För att se om ett kommando/skript har gått igenom utan error används .close(). Se nedan.

```
>>> output=os.popen('date')
>>> print(output.close())
None
```

En annan skillnad mellan os.system och os.popen kan man illustrera genom att pinga. I nedan pingas sidan ‘sunet.se’ med en -c flagga på 5 för att avslyta pinget efter 5 “packets”.

I det första exemplet används **os.system** och i nästa **os.popen**.

```
>>> os.system('ping -c 5 sunet.se')
PING sunet.se (37.156.192.51) 56(84) bytes of data.
64 bytes from sunet.se (37.156.192.51): icmp_seq=1 ttl=53 time=81.7 ms
64 bytes from sunet.se (37.156.192.51): icmp_seq=2 ttl=53 time=10.4 ms
64 bytes from sunet.se (37.156.192.51): icmp_seq=3 ttl=53 time=11.2 ms
64 bytes from sunet.se (37.156.192.51): icmp_seq=4 ttl=53 time=11.8 ms
64 bytes from sunet.se (37.156.192.51): icmp_seq=5 ttl=53 time=9.50 ms

--- sunet.se ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
rtt min/avg/max/mdev = 9.498/24.901/81.659/28.389 ms
0
>>>
```

os.popen:

```
>>> os.popen('ping -c 5 sunet.se')
<os._wrap_close object at 0x7f30713c2550>
>>>
```

När vi använder os.popen() ser vi inte utdata i Python-skalet omedelbart utan utdata buffras. Vi ser bara den slutliga utdata när ping-kommandot är klart och för att se det måste vi lägga till .read() .

```
>>> os.popen('ping -c 5 sunet.se').read()
'PING sunet.se (37.156.192.51) 56(84) bytes of data.\n64 bytes from sunet.se (37.156.192.51):
icmp_seq=1 ttl=53 time=8.48 ms\n64 bytes from sunet.se (37.156.192.51): icmp_seq=2 ttl=53
time=18.4 ms\n64 bytes from sunet.se (37.156.192.51): icmp_seq=3 ttl=53 time=14.5 ms\n64 bytes
from sunet.se (37.156.192.51): icmp_seq=4 ttl=53 time=17.2 ms\n64 bytes from sunet.se
(37.156.192.51): icmp_seq=5 ttl=53 time=16.1 ms\n\n--- sunet.se ping statistics ---\n5 packets
transmitted, 5 received, 0% packet loss, time 4004ms\nrtt min/avg/max/mdev =
8.482/14.934/18.413/3.475 ms\n'
>>>
```

Källor:
https://stackabuse.com/executing-shell-commands-with-python/
https://codefather.tech/blog/shell-command-python/

7. Inbyggda system eller web-serverskript

Inbyggda system:

Inbyggda funktioner: Python har ett antal inbyggda funktioner.De laddas automatiskt när ett skal startar och är alltid tillgängliga. Nedan exempel:

- print()
- input()
- int()
- list()

Inbyggda moduler: Det finns många inbyggda moduler i python som innehåller en lång rad värdefulla funktioner. Exempel på dessa moduler är:

- random. Modulen random används för att generera slumpmässiga siffror.
    - tex. random.randint

```
# Välj ett slumpmässigt värde mellan 1 och 1000
import random
print (random.randint(1,1000))
```

- math. Med math får du tillgång till vanliga matematiska konstanter.
    - tex math.sqrt()

```
import math
print(math.sqrt(16))  #kvadratrot av 16 = 4.0
```

- datetime. Innehåller funktioner som används för att hantera datum.
    - tex. time.time()

```
import datetime
import time
print(time.time())  # Returnerar antal sekudner sedan 1970-01-01
```

Inbyggd databas: Python har en inbyggd databas som heter SQLite. Jämförelse med andra databaser så har SQLite alla funktioner som behövs för en relationsdatabas med skillnaden att allt sparas i en enda fil. SQLite är inbyggt i ett Python-bibliotek vilket gör att du inte behöver installera någon programvara på serversidan/klientsidan. Några användningsområden som SQLite kan användas till är:

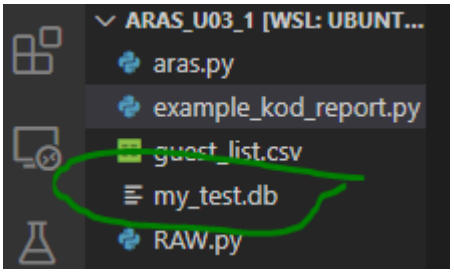
- Dataanalys
- Dataöverföring
- Interna eller tillfälliga databaser

Några steg som visar hur jag har använt SQLite kan ses nedan.

SQLite importeras enligt nedan.

```
import sqlite3 as sl # Import sqlite
con = sl.connect('my_test.db')# Create a data base connection
```

Nedan kan ses att ‘my_test.db’ vilket betyder att databasen är skapad och anslutningen fungerar.



Nedan ses följande steg:

- Skapande av en tabell
- Skapa värden i tabellen '**aras_family**'
- Infoga värden via den skapade anslutningen.
- Skapa frågor ur tabellen med ett resultat som kommer i output. Här har jag valt att visa alla i saras_family som är 16 år och äldre.

```
with con: # Create table
    con.execute("""
        CREATE TABLE ARAS_FAMILY (
            id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
            name TEXT,
            age INTEGER
        );
    """)
# Insert Records into the ARAS_FAMILY table we just created,
sql = 'INSERT INTO ARAS_FAMILY (id, name, age) values(?, ?, ?)'
data = [
    (1, 'Staffan', 53),
    (2, 'Simon', 18),
    (3, 'Anton', 16)
]

with con: #Insert the records with the connection object.
    con.executemany(sql, data)

with con: #Set the question to get from the table. Here: show the aras_family
# with the age of 16 and older.
    data = con.execute("SELECT * FROM ARAS_FAMILY WHERE age >= 16")
    for row in data:
        print(row)
```

output

```
(1, 'Staffan', 53)
(2, 'Simon', 18)
(3, 'Anton', 16)
```

Källor:
https://www.tutorialsteacher.com/python/python-builtin-modules
https://science.nu/courses/data-science-machine-learning-statistik-python/lektion/moduler-objekt-klasser-python/
https://towardsdatascience.com/do-you-know-python-has-a-built-in-database-d553989c87bd

Webbserver Scripting:
Serverskript är en teknik som används vid utveckling av webbsidor. Script ligger på en server och på användarens begäran till den specifika webbsidan så kopplas skripten in och ger ett kundanpassat svar.
Python har ett verktyg, Scrapy som effektivt extraherar (scrape) ut information från webbplatser. Att manuellt ta ut information från flera webbsidor är tidskrävande. I stället är lösningen att göra detta automatiskt. Det brukar kallas att man använder sig av “spindlar” eller “sökrobotar” som snabbt och effektivt söker igenom webbsidor efter information.

Efter installation av scrapy i terminalen kan det ses att scrapy version 1.7.3 är installerad. Se nedan bild.

```
sudo apt install python3-scrapy
```

```
aras@LAPTOP-LAR3R2HH:/mnt/c/Users/aras/u03_python_project/aras_u03_1$ scrapy -VScrapy 1.7.3 - no
active project
```

Ett projekt startas med kommando:

```
scrapy startproject aras_scrapy_gfg
```

```
aras@LAPTOP-LAR3R2HH:/mnt/c/Users/aras/u03_python_project/aras_u03_1$ scrapy startproject
aras_scrapy_gfg
```

Nedan kan ses att en mapp med ovan namn har skapats.

```
aras@LAPTOP-LAR3R2HH:/mnt/c/Users/aras/u03_python_project/aras_u03_1$ ll
total 28
drwxrwxrwx 1 aras aras  4096 Jan 15 19:18  ./
drwxrwxrwx 1 aras aras  4096 Jan 14 10:41  ../
-rwxrwxrwx 1 aras aras  2818 Jan 11 15:17  RAW.py*
-rwxrwxrwx 1 aras aras  3235 Jan 11 16:03  aras.py*
drwxrwxrwx 1 aras aras  4096 Jan 15 19:12  aras_scrapy_gfg/
-rwxrwxrwx 1 aras aras  2081 Jan 15 18:58  example_kod_report.py*
-rwxrwxrwx 1 aras aras    51 Jan 10 21:44  guest_list.csv*
-rwxrwxrwx 1 aras aras 12288 Jan 15 16:23  my_test.db*
-rwxrwxrwx 1 aras aras   464 Jan 12 11:58 'test av kod.py'*
-rwxrwxrwx 1 aras aras   398 Jan  8 16:38 'test kolumnutskrift.py'*
-rwxrwxrwx 1 aras aras   790 Jan 11 21:37  test_av_kod2.py*
```

Går man in i mappen kan nedan innehåll ses. Notera skapandet av mappen “**spiders”**.

```
aras@LAPTOP-
LAR3R2HH:/mnt/c/Users/aras/u03_python_project/aras_u03_1/aras_scrapy_gfg/aras_scrapy_gfg$ ll
total 8
drwxrwxrwx 1 aras aras 4096 Jan 15 18:57 ./
drwxrwxrwx 1 aras aras 4096 Jan 15 19:12 ../
-rwxrwxrwx 1 aras aras    0 Aug  1  2019 __init__.py*
drwxrwxrwx 1 aras aras 4096 Jan 15 18:57 __pycache__/
-rwxrwxrwx 1 aras aras  295 Aug  1  2019 items.py.tmpl*
-rwxrwxrwx 1 aras aras 3613 Aug  1  2019 middlewares.py.tmpl*
-rwxrwxrwx 1 aras aras  296 Aug  1  2019 pipelines.py.tmpl*
-rwxrwxrwx 1 aras aras 3162 Aug  1  2019 settings.py.tmpl*
drwxrwxrwx 1 aras aras 4096 Jan 15 18:57 spiders/
```

I mappen ‘spiders’ läggs en pyton fil där nedan skript läggs in. Notera att jag här lagt in webbadressen till ‘**klart.se’**.

```
class python_Spider(scrapy.Spider):
    name = "wether_klart"

    start_urls = [
        'https://www.klart.se/',
    ]

    def parse(self, response):

        for item in response.css('ol'):

            yield {
                'title': item.css('a::text').get(),
                'link': item.css('a::attr(href)').get(),
            }
```

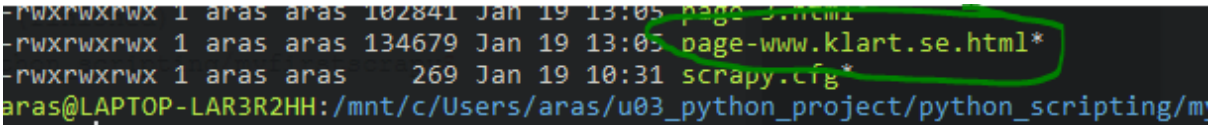
För att starta ‘scrape’ ställer man sig i projektets topp mapp och kör nedan kommando:

```
scrapy crawl wether_klart
```

Följande output ses(obs nedan visar bara slutdelen av körningen):

```
 'start_time': datetime.datetime(2022, 1, 19, 12, 5, 42, 401863)}
2022-01-19 13:05:43 [scrapy.core.engine] INFO: Spider closed (finished)
aras@LAPTOP-LAR3R2HH:/mnt/c/Users/aras/u03_python_project/python_scripting/myfirstscrapy$
```

I mappen kan nu ses att ett html fil har skapats med namn ‘page-www.klart.se.html



Källor:
https://www.geeksforgeeks.org/scraping-dynamic-content-using-python-scrapy/
https://www.simplifiedpython.net/scrapy-python-tutorial/
https://en.wikipedia.org/wiki/Server-side_scripting